Updated Tagging Design - Internal

This page contains implementation and internal details for the Updated Tagging Design.

Representations

Tags representation

- tag-key - unique identifier for a tag. Scoped by client portion
    - example: "org.nasa.something.quality"
- Description - optional string describing a tag
- originator-id - The username of the user who first created the tag.

Changes from previous

- Removed category, associated-concept-ids
- Combined namespace and value into tag-key

Tag Association Representation

A tag association connects a tag and another concept. Arbitrary data can be provided along with the association

- tag-key - The tag key
- associated-concept-id - the concept id of the associated concept
- revision-id - optional to indicate specific revision is tagged.
- Optional choice of one of these:
    - data - optional arbitrary data. Limited to 32K bytes of JSON max.
    - value - a string that will be indexed with the collection for searching.

Metadata DB Representation

Both are stored in metadata db as immutable revisioned concepts.

- Tags
    - Native-id: tag-key
    - Concept id type prefix: T
- Tag Association
    - native-id: tag-key + associated-concept-id + revision-id
        - Revision id may not be present. The native id is generated but never parsed. They will be separated by the group separator character so that the tag key can contain any other kinds of characters like / or comma. **Update**: This has been changed to a "/" because the group separator is not visible in tools like DB Visualizer, making things difficult.
    - Concept id type prefix: TA

Global transaction id

We will add a new single global transaction id to metadata db. It will be generated by selecting from a single Oracle sequence during insertion of any new revisions.

What problem is this solving?

This solves a problem with indexing collections with tag information. The current approach for finding collections by tag searches for tags first, gets the associated collection ids from the tag results and includes that in the query. The problem with this approach is there could be a very large number of collections associated with one tag. This would create a query that was too large and didn't perform well.

A better way to solve the collection search problem is to denormalize and index the associated tags with each collection. This will perform better both for finding collections and retrieving tag information with collection results.

If we index the elastic document as a collection denormalized with associated tags the document no longer represents the latest revision of a "collection". It represents a collective state of multiple documents. Because of this the single collection's revision id is not appropriate for use as the _version in elastic. The associations with a collection could change while the latest revision id of the collection is the same.

The global transaction id can be seen as the collective revision id for the entire database. Every change generates a new transaction id that is monotonically increasing.

Can you generate enough sequence values to represent every transaction?

There are 315.4 billion milliseconds in 10 years. The maximum oracle sequence value is 10^28. Long.MAX_VALUE isn't as big as the max sequence value but it's larger than the number of milliseconds in 10 years by several order of magnitudes. We could easily handle many concurrent operations per millisecond for centuries without exhausting the sequence size.

What does Elasticsearch allow for a max _version?

"Version numbers must be integers greater than zero and less than about 9.2e+18--a positive long value in Java"

Will Oracle have contention when using a single sequence for every transaction?

We'd like to use this approach for every revision including granules eventually. Based on this answer https://asktom.oracle.com/pls/asktom/f?p=100:11:::::P11_QUESTION_ID:2985886242221 it looks like there is overhead when using a single sequence versus a separate sequence for individual tables. This blog post http://www.pythian.com/blog/performance-issues-with-the-sequence-nextval-call/ has some good information about optimizing the sequences. It looks like we should set the sequence CACHE value to 100 instead of the default 20. We'll have to do performance testing with the change in place to see what the impact is to ingest.

Initial global transaction id

We use revision_id currently for version. For this change to work we must set global transaction id to start larger than the current largest revision id in any environment.

Preventing out of order global transaction ids

The best way to set the global transaction id would be to do it as part of the insert statement so that it happens as close to the time of insertion as possible. The problem here is that we might need to select the data back out of the database after it was inserted. If we select it out then insert that value network blips or other things could cause us to insert things in a different order than the global transaction ids were created. We want to avoid this.

There are a few different ways highlighted on these links below to set to the id during insert and retrieve it

- http://viralpatel.net/blogs/oracle-java-jdbc-get-primary-key-insert-sql/
- http://stackoverflow.com/questions/3552260/plsql-jdbc-how-to-get-last-row-id/3552353#3552353
    - May be out of date since previous link indicates it works.

Indexing

Three searching use cases

1. Search by tag key
2. Search by tag key with value
3. Include tag key with data/value in response

Collections are indexed with associated tags including tag key, tag value, and tag data.

The collection _version will be the global transaction id from the event that caused the reindexing to happen. This will mean that two events related to the same collection (like collection update, and tag association) from different tables could be indexed at the same time. We'll keep the data from the one with the more recent data.

Elasticsearch Mappings

These are some initial ideas for Elasticsearch mappings

Add tag assocations to collections

```
:tag-associations tag-associations-mapping
```

The nested mapping for tag associations:

```
(def tag-associations-mapping
  {:type "nested"
   :dynamic "strict"
   :_source {:enabled false}
   :_all {:enabled false}
   :properties {:tag-key (stored string-field-mapping)

                ;; Revision id and date are from the tag association so we can return the date that collection was last associated

:revision-id (not-indexed (stored string-field-mapping))

:revision-date (not-indexed (stored date-field-mapping))

                :originator-id string-field-mapping

:value (stored string-field-mapping)
                :data-gzip-b64 (not-indexed (stored string-field-mapping))}})
```

Tags will still be indexed but probably look like this

```
(def tag-mapping
  {:tag
   {:dynamic "strict",
    :_source {:enabled false},
    :_all {:enabled false},
    :_id  {:path "concept-id"},
    :_ttl {:enabled true},
    :properties {:concept-id (stored string-field-mapping)
                 :originator-id (stored string-field-mapping)

:tag-key (stored string-field-mapping)
                 :tag-key.lowercase string-field-mapping
                 :description (not-indexed (stored string-field-mapping))}}})
```

This removes namespace, category, value, and the associated concept ids.

Finding tags associated with a collection during indexing

We'll find every tag association currently associated with the collection. This must be found by searching against metadata db. We can't search elastic since it's eventually consistent.

We'll send a request to metadata db to search for tag associations with the following parameters

- associated-concept-id
    - The collection's concept id
- latest=true
    - Only return the latest revisions of those associations. Associated concept id and the associated revision can't change from revision to revision so the latest would be sufficient.

This will the latest revision of the tag associations that are associated with that concept (or a revision of the concept). If the revision is present we'll check that it matches the collection revision.

Note that we're not including the global transaction id in the parameters that are sent when finding tags associated with a collection. We don't need to do this because we're really only interested in indexing whatever the current associations are in elasticsearch. We don't have a usecase yet for getting the associations with a collection from a certain point in the past.

Reindexing collections

When reindexing collections we can't use the global transaction id from the collection table. Those transaction ids would be in the past and there might be other events that had surpassed them. We need to use a global transaction id of *now*. Metadata DB needs to support a REST API for retrieving the current value of the global transaction id sequence. That would be the transaction id last used. The reindexing process will fetch that at the start of reindexing as the collection _version.

Update

There is no reliable way of retrieving the last used transaction id without explicitly storing it somewhere. Oracle fetches multiple sequence values at once and caches them. When you ask for the highest sequence value, it gives you the highest cached value, not the highest *used* value. So instead of using the highest global transaction id in the database, we are using the highest transaction id of the the set of transaction ids belonging to the collection (latest revision) and the latest revisions of the tag associations for the collection.

Collection revision cleanup/force delete

Metadata DB sends an event when old revisions are removed either during cleanup or force deletes. We need to do the following things when that happens.

- Remove tag associations with that specific revision (handled in metadata db)
    - These should be removed completely. We no longer need the info kept around.

Processing Tag Association Events

When processing a message of an updated tag association we'll reindex the related collection. When we do this we'll use the global transaction id of the tag association. This will become the _version used within elasticsearch.

Update

We are using the *highest* transaction id between the collection and all tag associations for _version, as described above in **Reindexing Collections**.

Tagging ACLs

We would like to be able to grant clients the ability to manipulate their own tags but not manipulate the tags of other clients. We can do this by creating a new ACL type that identifies a prefix when granting access for writing ACLs. For example the EDSC user would be granted permission to update tags with "gov.nasa.earthdata.search.". EDSC would not be able to modify tags owned by GCMD which would have a different prefix. We'll need to wait until we transition ACLS to the CMR before we make this change.

Metadata DB and Indexed data walkthrough

Key for shorter acronyms for smaller tables below.

- rid = revision id
- gti = global transaction id
- d = deleted

New rows are flagged with *

Initial database state

A few collections exist but no tags. Tags table and associations are empty

PROV1 Collections table

concept-id	rid	d	gti
C1-PROV1	1		1
C1-PROV1	2		2
C2-PROV1	1		3

Tags Table (empty)

concept-id	rid	d	gti

Tag Associations Table (empty)

concept-id	rid	d	gti

Latest Collections Index

concept-id	revision-id	_version	tags
C1-PROV1	2	2	
C2-PROV1	1	3	

All revision index

concept-id	rid	_version	tags
C1-PROV1	1	1	
C1-PROV1	2	2	
C2-PROV1	1	3	

A collection is associated with a tag

EDSC assocated C1-PROV1 with in_modaps

PROV1 Collections table (no change)

concept-id	rid	d	gti
C1-PROV1	1		1
C1-PROV1	2		2
C2-PROV1	1		3

Tags Table

concept-id	rid	d	gti	tag-key
*T1-CMR	1		4	in_modaps

Tag Associations Table

concept-id	rid	d	gti	assoc concept id	assoc rid
*TA1-CMR	1		5	C1-PROV1	

Latest Collections Index

concept-id	rid	_version	tags
*C1-PROV1	2	5	in_modaps
C2-PROV1	1	3	

All revision index

Note all revisions of a collection in the all revisions indexed have to be reindexed.

concept-id	rid	_version	tags
*C1-PROV1	1	5	in_modaps
*C1-PROV1	2	5	in_modaps
C2-PROV1	1	3	

A collection revision is associated with a tag

GCMD associates C1-PROV1 revision 1 with tag review_status with value IN_REVIEW

PROV1 Collections table (no change)

concept-id	rid	d	gti
C1-PROV1	1		1
C1-PROV1	2		2
C2-PROV1	1		3

Tags Table

concept-id	rid	d	gti	tag-key

T1-CMR	1		4	in_modaps
*T2-CMR	1		6	review_status

Tag Associations Table

concept-id	rid	d	gti	assoc concept id	assoc rid
TA1-CMR	1		5	C1-PROV1	
*TA2-CMR	1		7	C1-PROV1	1

Collection Elastic Search Collection Index

concept-id	rid	_version	tags
C1-PROV1	2	5	in_modaps
C2-PROV1	1	3	

All revision index

concept-id	rid	_version	tags
*C1-PROV1	1	7	in_modaps, review_status=IN_REVIEW
C1-PROV1	2	5	in_modaps
C2-PROV1	1	3	

Collection dissassociated

EDSC dissassociated C1-PROV1 from in_modaps

PROV1 Collections table (no change)

concept-id	rid	d	gti
C1-PROV1	1		1
C1-PROV1	2		2
C2-PROV1	1		3

Tags Table

concept-id	rid	d	gti	tag-key
T1-CMR	1		4	in_modaps
T2-CMR	1		6	review_status

Tag Associations Table

concept-id	rid	d	gti	assoc concept id	assoc rid
TA1-CMR	1		5	C1-PROV1	
*TA1-CMR	2	true	8	C1-PROV1	
TA2-CMR	1		7	C1-PROV1	1

Collection Elastic Search Collection Index

concept-id	rid	_version	tags
*C1-PROV1	2	8	
C2-PROV1	1	3	

All revision index

concept-id	rid	_version	tags

*C1-PROV1	1	8	review_status=IN_REVIEW
*C1-PROV1	2	8	
C2-PROV1	1	3	

Tag deleted

GCMD deletes the review_status tag

PROV1 Collections table (no change)

concept-id	rid	d	gti
C1-PROV1	1		1
C1-PROV1	2		2
C2-PROV1	1		3

Tags Table

concept-id	rid	d	gti	tag-key
T1-CMR	1		4	in_modaps
T2-CMR	1		6	review_status
*T2-CMR	2	true	9	review_status

Tag Associations Table

concept-id	rid	d	gti	assoc concept id	assoc rid
TA1-CMR	1		5	C1-PROV1	
TA1-CMR	2	true	8	C1-PROV1	
TA2-CMR	1		7	C1-PROV1	1
*TA2-CMR	2	true	10	C1-PROV1	1

Collection Elastic Search Collection Index

concept-id	rid	_version	tags
C1-PROV1	2	8	
C2-PROV1	1	3	

All revision index

concept-id	rid	_version	tags
*C1-PROV1	1	10	
C1-PROV1	2	8	
C2-PROV1	1	3	

Questions

How will it work if someone deletes a tag and then a new tag association is created?

This is similar to collection deletion while a granule insert is coming in. We want the deletion of a tag to cascade to deletion of all the existing associations. But we don't want an immediate subsequent tag creation and association to end up with the wrong thing.

Answer:

- Create the tag deletion revision
- Capture transaction id of deletion
- Asynchronously find and remove all tag associations that have a transaction id less than the captured one.

Anything that was tagged after this point would have a larger transaction id and should be skipped.

What happens if someone tags something and the tag doesn't exist?

We will create the tag if it doesn't exist.

Scaling question

What happens when there are many collections associated with a single tag?

There will be as many tag associations in Metadata DB as there are collections. Each collection would be indexed with the tag information. There shouldn't be any issues with this.

What happens when there are many revisions associated with a single tag? Can we support tagging every revision?

This is the same as the previous answer. The design should be able to handle many revisions. We can test tagging every revision of all collections in workload if needed.

What happens when there are a lot of associations with data?

We'll have to store that data. 32K max data size * 50,000 collections * 10 revisions would be 16GB used if a client developer created a tag with max data size and tagged every possible collection revision. That's a reasonable amount of data that we could support in the worst case.

Note that it's unlikely they'd use that much data. We don't actually have 10 revisions of every collection.

We also store the data compressed in Elasticsearch and in Metadata DB so we'd actually store much less than this.

What happens when there are many tags?

Many tags would correlate to many tags in Metadata DB which isn't a problem.

What happens if there are many tags associated with a single collection?

This is a limitation of the current design. All of the tags associated with a collection are stored in the collection. There's a limit on the amount of data elasticsearch could reasonable put in one document. It would also potentially create very large response sizes.

We should consider limiting the number of tags a client could create. They should be able to have any many associations as they want. The current tagging design should be flexible enough for clients that they don't need very many tags. A reasonable maximum might be between 50 and 200 tags per client.

What should we do if a concept is tagged and they try to tag a specific revision or vice versa?

We can't allow both a single revision and the larger concept to be tagged at the same time. It doesn't make sense to do this. If both of them had attached data which one would be returned in search results? If a client is doing this it's most likely an error. Tags will most likely be used for specific revisions or whole collections but not both.

- If a collection is already tagged with a specific revision we should reject an attempt to tag the entire collection without a revision.
- If a collection is already tagged without a revision we should reject an attempt to tag that a single rev